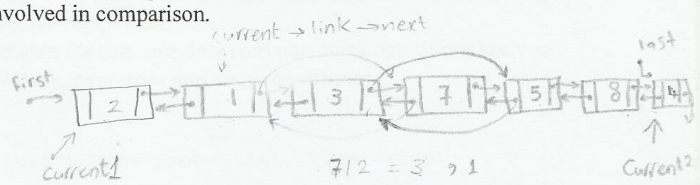
Question 1 [9 Marks]

Write a *member function* called **compareHalf** to be included in class **doublyLinkedList**. If all the elements of the first half of the list are less than all the elements of the second half of the list, then the function will return true, else it will return false. If the list is empty or has only one element, return true. If the number of elements in the list is odd, then the middle element is not involved in comparison.

Function prototype:

```
bool compareHalf();
```



Example:

List: 2 1 3 7 5 8 4

Note that all the elements of the first half, i.e., 2 1 3 are less than all the elements of the second half, i.e., 5 8 4. Therefore, the function will return true. Also note that the middle element 7 is not involved in comparison.

Assume that the class **doublyLinkedList** contains following private data members:

```
nodeType<Type> *first;     // pointer to the first node
nodeType<Type> *last;      // pointer to the last node
int count;                 // number of nodes
```

```
bool doublyLinkedList<Type>::compareHalf()
{
  if (first == NULL || first->link == NULL)
  {
    return true;
  }
  if (count % 2 != 0) // odd
  {
    doublyLinkedList<Type> *current1, *current2;
    current1 = first;
    current2 = last;
    for (int i = first; i < count/2; i++)
    for (int j = count; i < count/2
                        ; j--)
    {
      if (current1->info > current2->info)
        return true; break;
      else current1 = current1->link->next;
           current2 = current2->link->back;
    }
    return false;
  }
}
```

6

Question 2 [9 + 5 Marks]

(A) [9 Marks] Write a non-member function called **countAndDeleteKey** that accepts an object **st1** type **stackType** as the first parameter and **key** of type **Type** as the second parameter. The function will count the number of occurrences of **key** as **info** in **st1** and will return this count. The function will also delete all occurrences of **key** from **st1**. All the remaining elements of **st1** should be in the original relative order.
Assume that class **stackType** is available for use. Use only common stack operations such as push, pop, top, isEmptyStack, isFullStack, operator= and copy constructor.

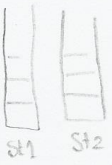
Function prototype:

```
int   countAndDeleteKey(stackType<Type>  &st1, Type  &key);
```

Example:
key = **5**

Stack st1 before function call: 10 **5** 12 15 **5** 10 20 **5** 30 2 20
top

Stack st1 after function call: 10 12 15 10 20 30 2 20
top



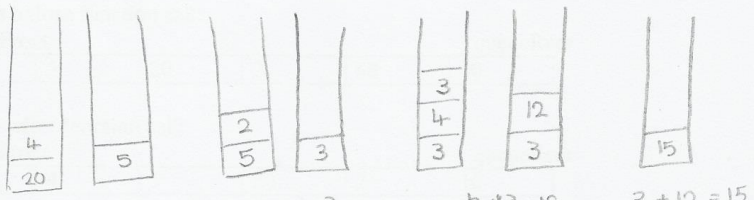
As **5** (key) occurs 3 times in st1, the function will **return 3**.

```
int StackType <Type> :: countAndDeleteKey (StackType <Type> &St1, Type &Key)
{
  StackType <Type> St2;
  int counter = 0; item;
  StackType <Type> St1 (St2);   St2(St1);
  while (St1.isEmptyStack())
  {
    St1.Pop();   +1
  }
  while (St2.isEmptyStack())
  {
    if (St2.top() != key)
    {
      item = St2.top();
      St1.Push(item);
      St2.Pop();
    }
    else
    {
      counter++;
      St2.Pop();
    }
  }
  return counter;
}
```

5 1/2

(B) [5 Marks] Consider the following <u>postfix expression</u>. Use stack to evaluate it and show all the push and pop operations by clearly drawing the stack status.

20 4 / 2 - 4 3 * +



not enough details.

5

Question 3 [9 Marks]

Write a member function **highestFirst** to be included in class **queueType** without having any parameters. If the queue is not empty, then the function will find the maximum element in the queue and will make it as the front element of the queue, the order of other elements in the queue will remain unchanged and will return true. If the queue is empty, the function will return false.

Example:

Queue before function call:

queueFront					queueRear
4	5	20	2	**60**	10

Queue after function call:

queueFront					queueRear
60	4	5	20	2	10

Function prototype:

```
bool  highestFirst( );
```

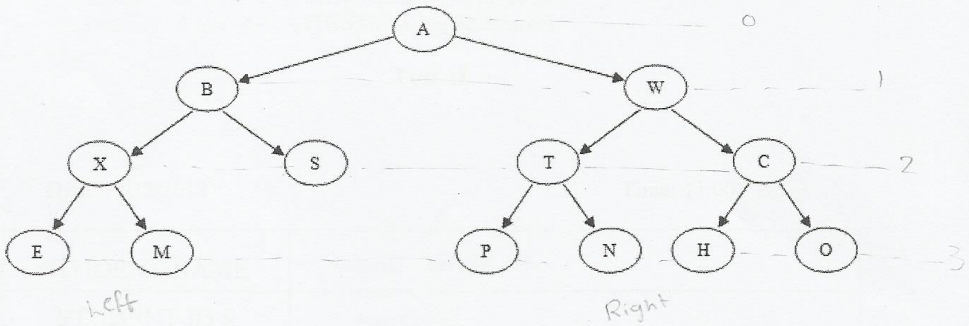
You may use common queue operations such as addQueue, deleteQueue, front, back, isEmptyQueue, isFullQueue, operator= and copy constructor in your function.

```
template <classType>
bool queueType <Type> :: highestFirst ()
{
  if (isEmptyQueue())
    return false;

  else
  {
    max = 0;
    for (int i = array[queueFront]; i < array[queueRear]; i++)
    {
      if (array[i] > max)
        max = array[i];
    }

    while (! isEmptyQueue())
    {
      int index
      index = (index+1) % maxQueueSize;
      queueFront = max;
    }
    return true;
  }
```

2½

Question 4 [8 Marks]

(A) For the binary tree given below, answer the following questions:



i. **[1 Marks]** What is the **height** of this binary tree?

4 = max level + 1

ii. **[2 Marks]** List all the leaf nodes in the right sub-tree of this binary tree.

O H N P

iii. **[5 Marks]** List the sequence of nodes, if the binary tree is traversed using **in-order traversal**.

(LDR)

EXMBSA PTN W HCO

8